**18EE504**

Hall Ticket Number:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |

**III/IV B.Tech (Regular\Supplementary) DEGREE EXAMINATION**

**Feb, 2021** **Electrical and Electronics Engineering**

**Fifth Semester** **Microprocessors and Microcontrollers**

**Time:** Three Hours **Maximum:** 50 Marks

*Answer all questions from Part-A.* (1X10= 10 Marks)
*Answer ANY FOUR questions form Part-B.* (4X10=40 Marks)

***Part-A***

| | | | |
|---|---|---|---|
| 1. | | Answer all questions | (1X10=10 Marks) |
| | a) | What is the maximum capacity of the memory that can be interfaced to the microprocessor which contains 20 address lines?<br>**1 MB** | C01 |
| | b) | What will be 20-bit physical address form which the code is accessed? If [CS]=348AH, [IP]=4214H.<br>**38AB4** | CO1 |
| | c) | What will be the contents of DL after execution of the instruction ROL DL,01H? If [DL]=EFH and carry flag is reset.<br>**DFH** | CO1 |
| | d) | Which of the modes of 8254 timer generates square wave?<br>**Mode 3** | C02 |
| | e) | In order to program Port A and Port B as output ports in Mode 0 and Port C as input port in Mode 0, the control word for 8255 would be<br>**89H** | CO2 |
| | f) | The 8051 has ____**2**____ 16-bit counter/timers. | CO3 |
| | g) | Find the number of times the following loop will be executed MOV R6,#100<br>BACK:MOV R5,#100<br>HERE:DJNZ R5,HERE<br>DJNZ R6,BACK<br>END<br>**10000** | CO4 |
| | h) | What is the total amount of external code memory that can be interfaced to the 8051?<br>**64 KB** | CO3 |
| | i) | Name two 16-bit registers of 8051 microcontroller **DPTR and PC** | CO3 |
| | j) | What is the functionality of ULN2003 IC in interfacing stepper motor with 8051 microcontrollers<br>**It acts as high current driver for stepper motor** | CO4 |
| | | **Part-B**<br>**UNIT-I** | | |
| 2. | a) | Draw the architecture of 8086 Microprocessor and Explain the functionality of each block. CO1 | 5M |
| | b) | Explain about different types of Addressing modes in 8086 with examples CO1 | 5M |
| | | | |
| 3. | a) | Explain about following instructions. (a) OUT (b) MUL (c) CMP (d) MOVSB CO1 | 5M |
| | b) | Explain about program development tools used in assembly language programming CO1 | 5M |
| | | **UNIT-II** | |
| 4. | a) | Explain about 8255 Programmable peripheral interfacing device with a neat sketch CO2 | 5M |
| | b) | Explain different modes in which 8255 is operated in connecting peripheral devices to 8086 CO2 | 5M |
| | | | |
| 5. | a) | Explain how a matrix keyboard is interfaced with 8086 microprocessors with the help of a neat sketch. CO2 | 5M |
| | b) | Write in detail about modes of operation of 8254 CO2 | 5M |
| | | **UNIT-III** | |
| 6. | a) | Write about the comparison of microprocessors with microcontrollers CO3 | 5M |
| | b) | Draw the internal architecture of 8051 and explain different blocks. CO3 | 5M |
| | | | |
| 7. | a) | Explain about memory organization of 8051 microcontrollers CO3 | 5M |
| | b) | Explain about I/O port programming in 8051 microcontrollers CO3 | 5M |
| | | **UNIT-IV** | |
| 8. | a) | Explain about interfacing of ADC with 8051 microcontrollers using a diagram CO4 | 5M |

|  |  |  |  |  |
|---|---|---|---|---|
|  | b) | Explain about timer port programming in 8051 microcontrollers | CO4 | 5M |
|  |  |  |  |  |
| 9. | a) | Explain how a stepper motor is interfaced to an 8051 μc with a neat diagram. | CO4 | 5M |
|  | b) | Draw the format of (i) TMOD (ii) SCON special function registers in 8051 microcontrollers. | CO4 | 5M |

## Part- B

|  |  |  |  |  |
|---|---|---|---|---|
| 2. | a) | Draw the architecture of 8086 Microprocessor and Explain the functionality of each block. | CO1 | 5M |

A **Microprocessor** is an Integrated Circuit with all the functions of a CPU however, it cannot be used stand alone since unlike a microcontroller it *has no memory or peripherals*.

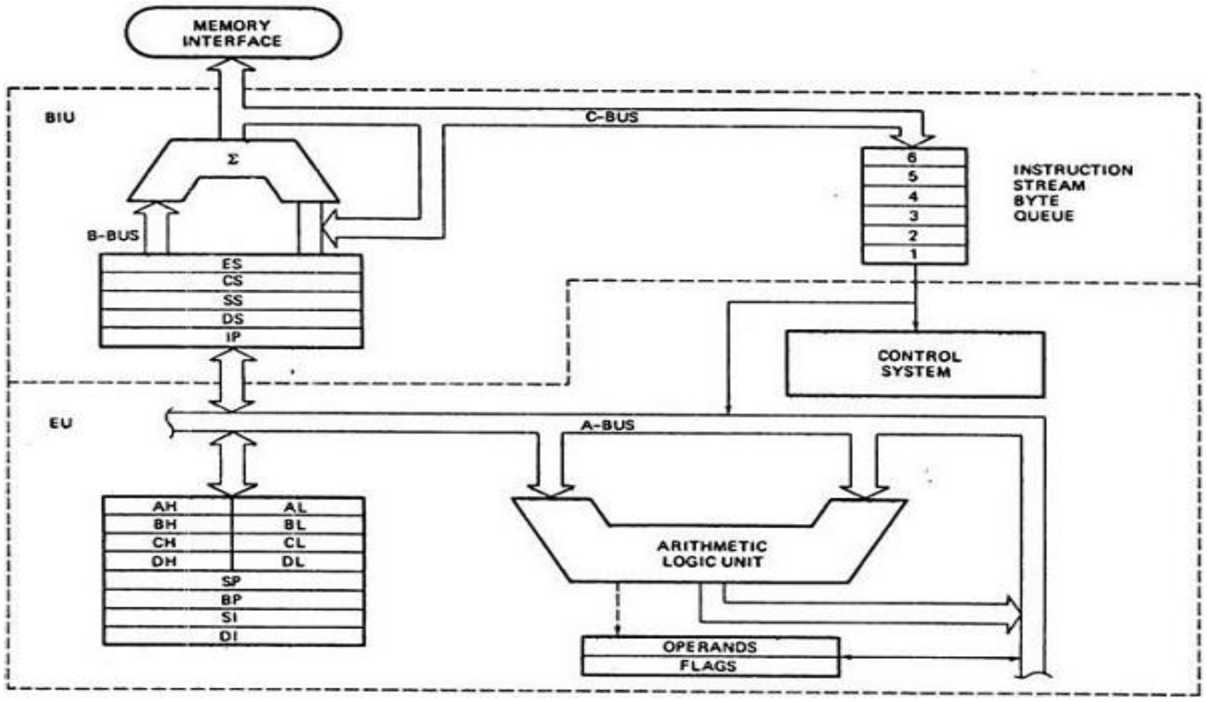
8086 does not have a RAM or ROM inside it. However, it has *internal registers* for storing intermediate and final results and interfaces with memory located outside it through the System Bus.
In case of 8086, it is a 16-bit **Integer processor** in a 40 pin, Dual Inline Packaged IC.
The size of the internal registers(present within the chip) indicate how much information the processor can operate on at a time (*in this case 16-bit registers*) and how it moves data around internally within the chip, sometimes also referred to as the internal data bus.
8086 provides the programmer with 14 internal registers, each 16 bits or 2 Bytes wide.
**1. The Bus Interface Unit (BIU):**
It provides the interface of 8086 to external memory and I/O devices via the System Bus. It performs various machine cycles such as memory read, I/O read etc. to transfer data between memory and I/O devices.
BIU performs the following functions-

- It generates the 20 bit physical address for memory access.
- It fetches instructions from the memory.
- It transfers data to and from the memory and I/O.
- Maintains the 6 byte prefetch instruction queue(**supports pipelining**).

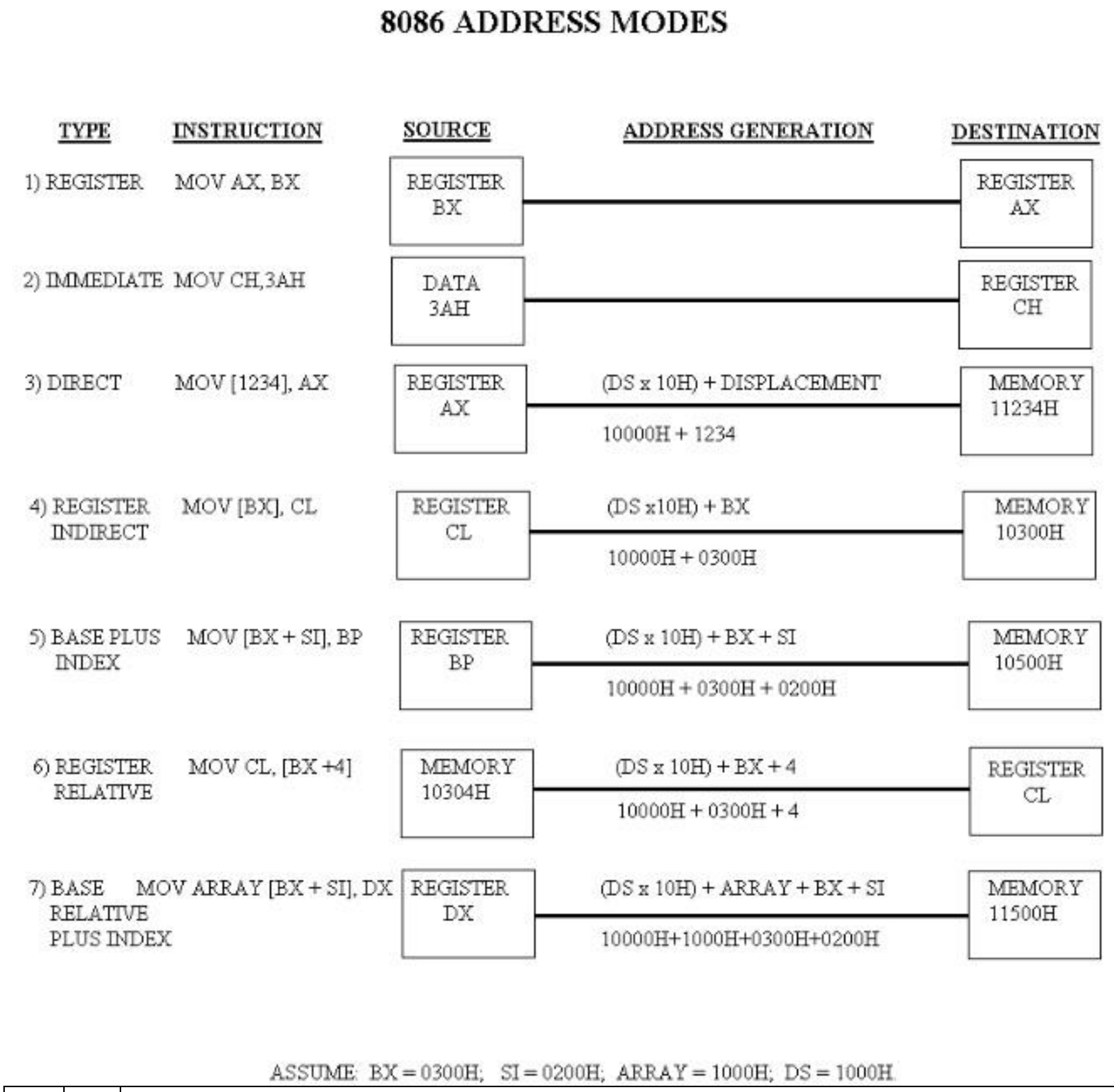
BIU mainly contains the **4 Segment registers**, the **Instruction Pointer**, a prefetch queue and an **Address Generation Circuit**.
**2. The Execution Unit (EU):**
The main components of the EU are General purpose registers, the ALU, Special purpose registers, Instruction Register and Instruction Decoder and the Flag/Status Register.

1. Fetches instructions from the Queue in BIU, decodes and executes arithmetic and logic operations using the ALU.
2. Sends control signals for internal data transfer operations within the microprocessor.
3. Sends request signals to the BIU to access the external module.
4. It operates with respect to T-states (clock cycles) and not machine cycles.

8086 has four 16 bit general purpose registers AX, BX, CX and DX. Store intermediate values during execution. Each of these have two 8 bit parts (higher and lower).



| | | | | |
|---|---|---|---|---|
| 2 | b) | Explain about different types of Addressing modes in 8086 with examples | CO1 | 5M |



**8086 ADDRESS MODES**

| TYPE | INSTRUCTION | SOURCE | ADDRESS GENERATION | DESTINATION |
|---|---|---|---|---|
| 1) REGISTER | MOV AX, BX | REGISTER BX | | REGISTER AX |
| 2) IMMEDIATE | MOV CH,3AH | DATA 3AH | | REGISTER CH |
| 3) DIRECT | MOV [1234], AX | REGISTER AX | (DS x 10H) + DISPLACEMENT<br>10000H + 1234 | MEMORY 11234H |
| 4) REGISTER INDIRECT | MOV [BX], CL | REGISTER CL | (DS x10H) + BX<br>10000H + 0300H | MEMORY 10300H |
| 5) BASE PLUS INDEX | MOV [BX + SI], BP | REGISTER BP | (DS x 10H) + BX + SI<br>10000H + 0300H + 0200H | MEMORY 10500H |
| 6) REGISTER RELATIVE | MOV CL, [BX +4] | MEMORY 10304H | (DS x 10H) + BX + 4<br>10000H + 0300H + 4 | REGISTER CL |
| 7) BASE RELATIVE PLUS INDEX | MOV ARRAY [BX + SI], DX | REGISTER DX | (DS x 10H) + ARRAY + BX + SI<br>10000H+1000H+0300H+0200H | MEMORY 11500H |

ASSUME: BX = 0300H; SI = 0200H; ARRAY = 1000H; DS = 1000H

| | | | | |
|---|---|---|---|---|
| 3. | a) | Explain about following instructions. (a) OUT (b) MUL (c) CMP (d) MOVSB | CO1 | 5M |

(a) `OUT im.byte, AL`
```
    im.byte, AX
    DX, AL
    DX, AX
```
Output from **AL** or **AX** to port.
First operand is a port number. If required to access port number over 255 - **DX** register should be used.

Example:
```
MOV AX, 0FFFh ; Turn on all
OUT 4, AX ; traffic lights.
MOV AL, 100b ; Turn on the third
OUT 7, AL ; magnet of the stepper-motor.
C Z S O P A
Unchanged
```
(b) `MUL REG`
```
    memory
```
Unsigned multiply.
Algorithm:
when operand is a **byte**:
`AX = AL * operand.`
when operand is a **word**:
`(DX AX) = AX * operand.`
Example:
```
MOV AL, 200 ; AL = 0C8h
MOV BL, 4
MUL BL ; AX = 0320h (800)
```

```
RET
```
CF=OF=0 when high section of the result is zero.

```
C Z S O P A
r ? ? r ? ?
```

```
(c) CMP   REG, memory
          memory, REG
          REG, REG
          memory, immediate
          REG, immediate
```
Compare.

Algorithm:

```
          operand1 - operand2
```
result is not stored anywhere, flags are set (OF, SF, ZF, AF, PF, CF) according to result.

Example:

```
MOV AL, 5
MOV BL, 5
CMP AL, BL ; AL = 5, ZF = 1 (so equal!)
RET
```

```
C Z S O P A
r r r r r r
```

(d) MOVSB No operands

Copy byte at DS:[SI] to ES:[DI]. Update SI and DI.

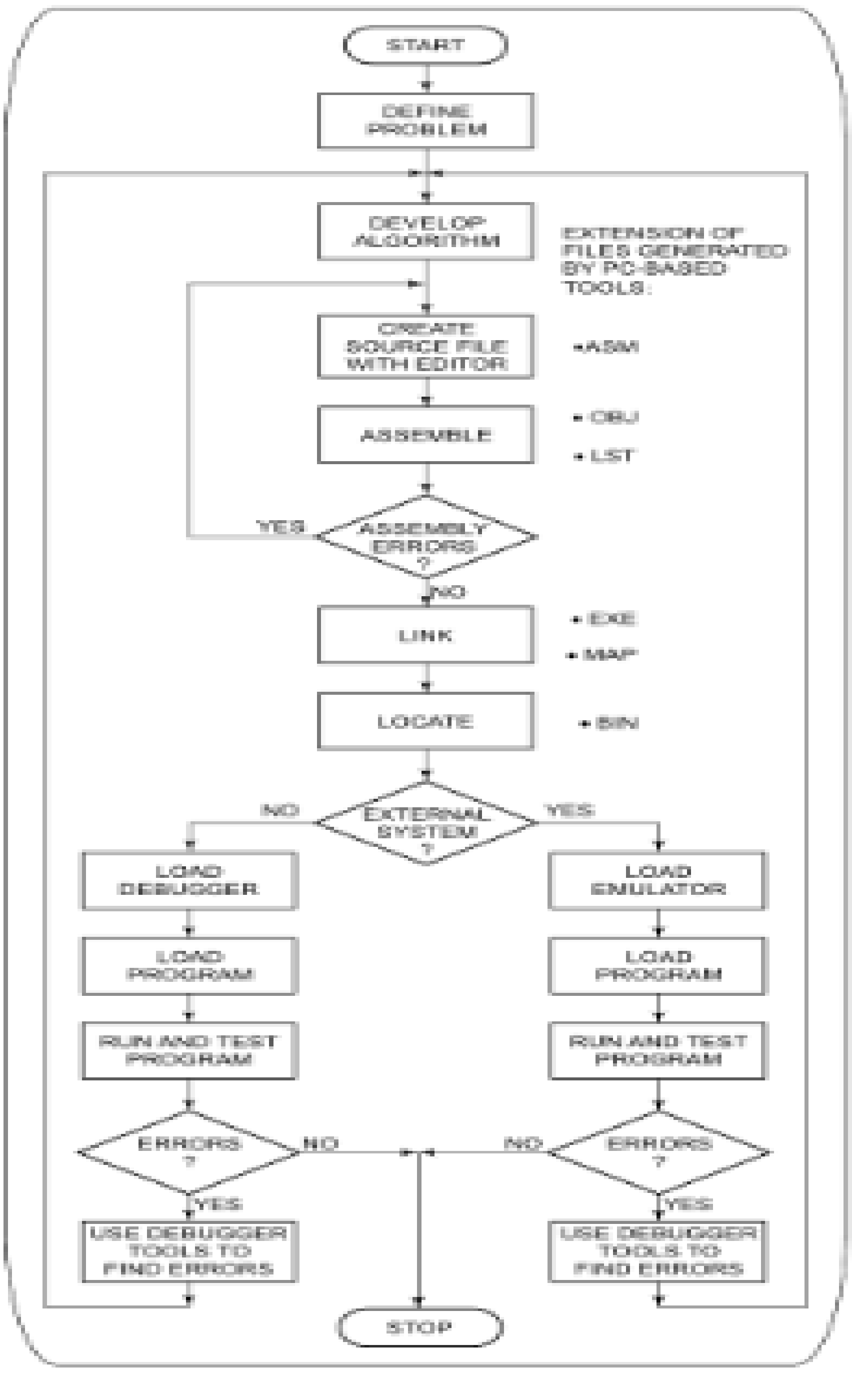
Algorithm:

```
▯ ES:[DI] = DS:[SI]
▯ if DF = 0 then
▯ SI = SI + 1
▯ DI = DI + 1
else
▯ SI = SI - 1
▯ DI = DI - 1
```
Example:

```
ORG 100h
CLD
LEA SI, a1
LEA DI, a2
MOV CX, 5
REP MOVSB
RET
a1 DB 1,2,3,4,5
a2 DB 5 DUP(0)
```

```
C Z S O P A
Unchanged
```

| | | | | |
|---|---|---|---|---|
| 3 | b) | Explain about program development tools used in assembly language programming | CO1 | 5M |



Program development algorithm

| | | | | |
|---|---|---|---|---|
| 4. | a) | Explain about 8255 Programmable peripheral interfacing device with a neat sketch | CO2 | 5M |

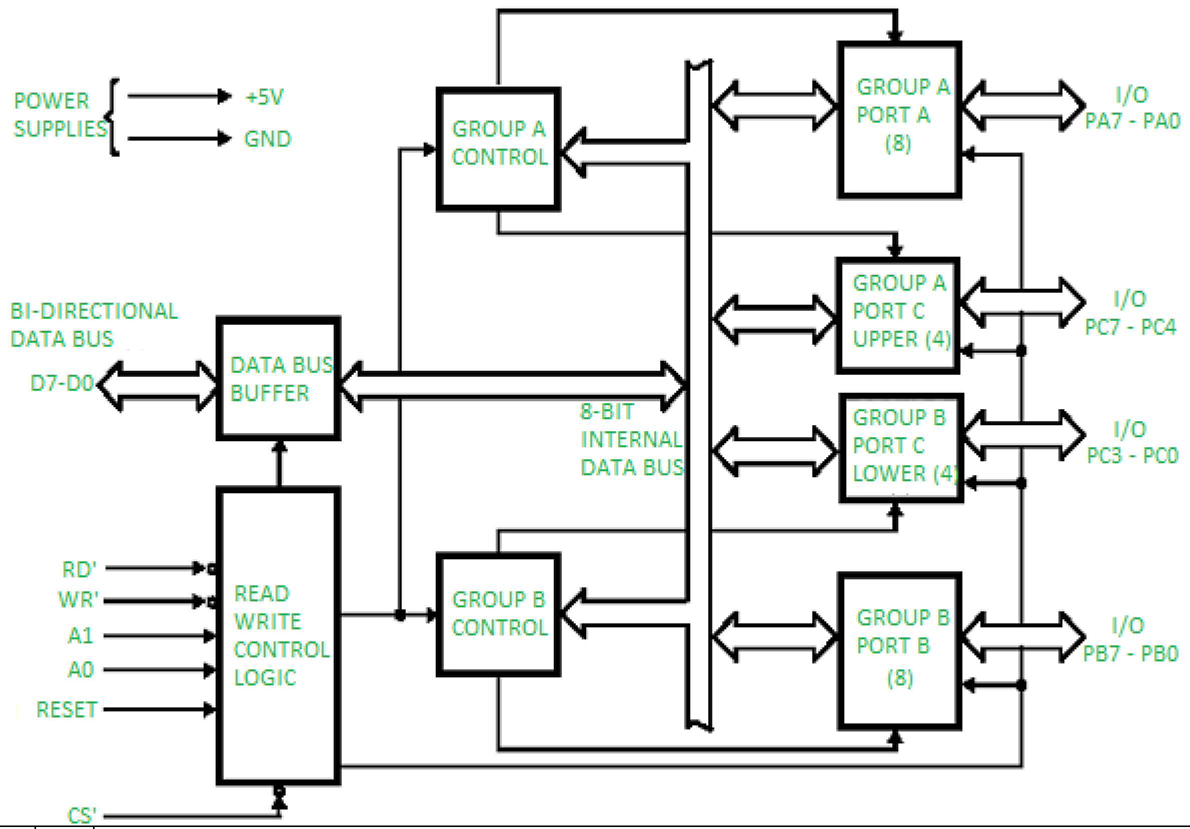
PPI 8255 is a general purpose programmable I/O device designed to interface the CPU with its outside world such as ADC, DAC, keyboard etc. We can program it according to the given condition. It can be used with almost any microprocessor.

It consists of three 8-bit bidirectional I/O ports i.e. PORT A, PORT B and PORT C. We can assign different ports as input or output functions.

It has two control groups, control group A and control group B. Control group A consist of port A and port C upper. Control group B consists of port C lower and port B.

Depending upon the value if CS', A1 and A0 we can select different ports in different modes as input-output function or BSR. This is done by writing a suitable word in control register (control word D0-D7).

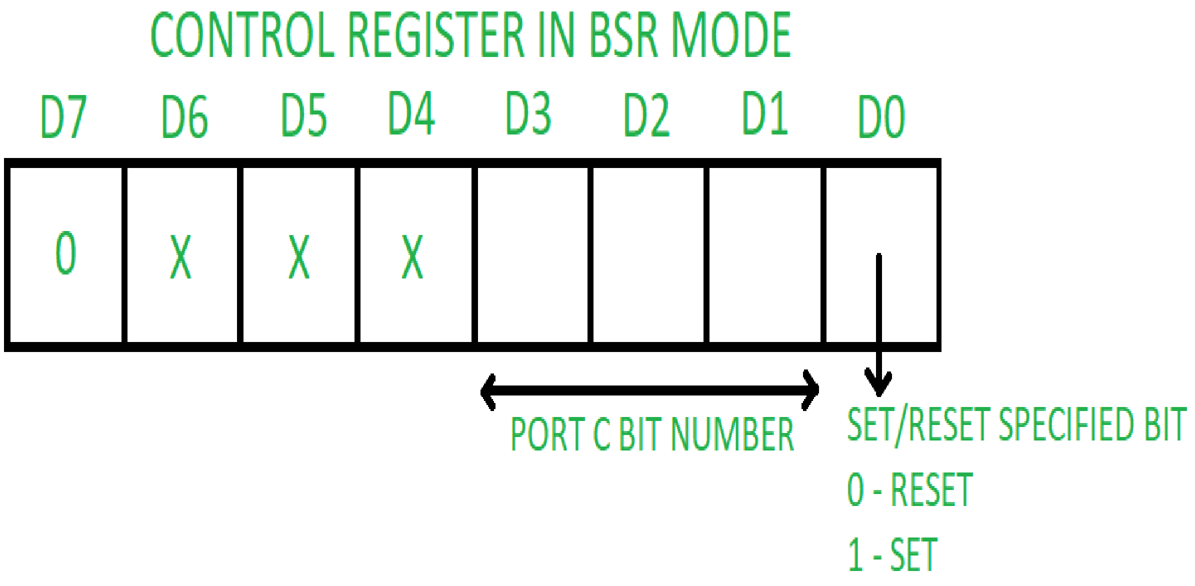
| CS’ | A1 | A0 | Selection | Address |
|---|---|---|---|---|
| 0 | 0 | 0 | PORT A | 80 H |
| 0 | 0 | 1 | PORT B | 81 H |
| 0 | 1 | 0 | PORT C | 82 H |
| 0 | 1 | 1 | Control Register | 83 H |
| 1 | X | X | No Seletion | X |



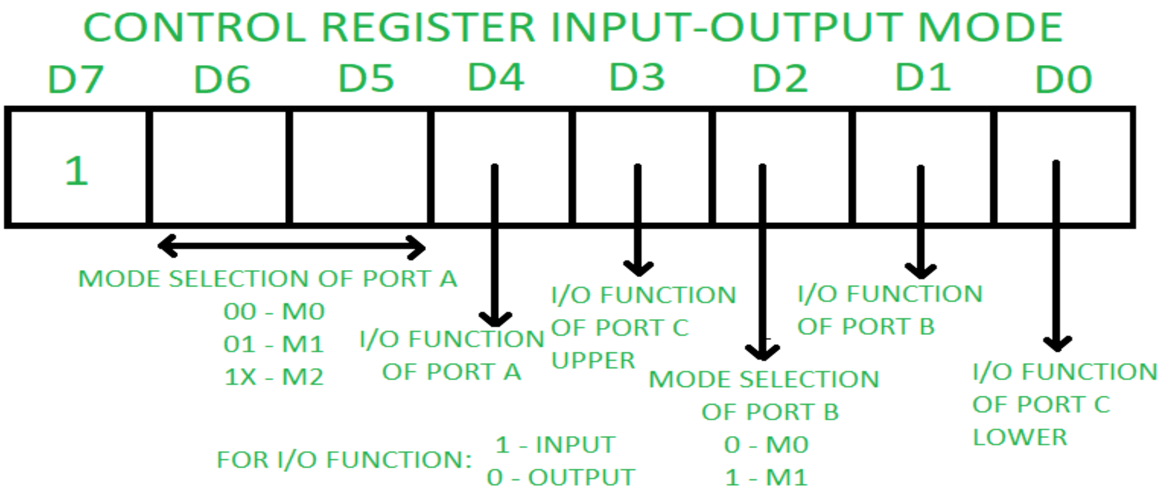
|   |   |   |   |   |
|---|---|---|---|---|
| 4 | b) | Explain different modes in which 8255 is operated in connecting peripheral devices to 8086 | CO2 | 5M |

**Operating modes –**

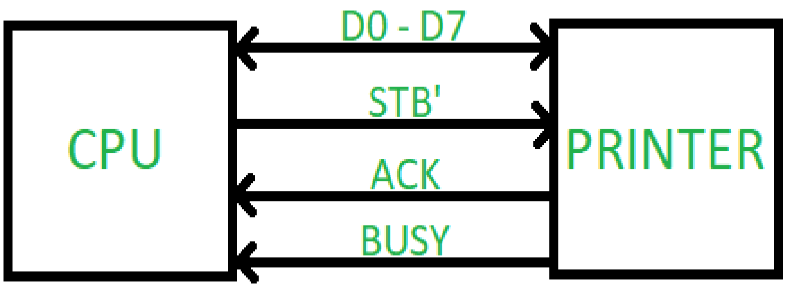
1. **Bit set reset (BSR) mode –**
   If MSB of control word (D7) is 0, PPI works in BSR mode. In this mode only port C bits are used for set or reset.



2. **Input-Output mode –**
   If MSB of control word (D7) is 1, PPI works in input-output mode. This is further divided into three modes:

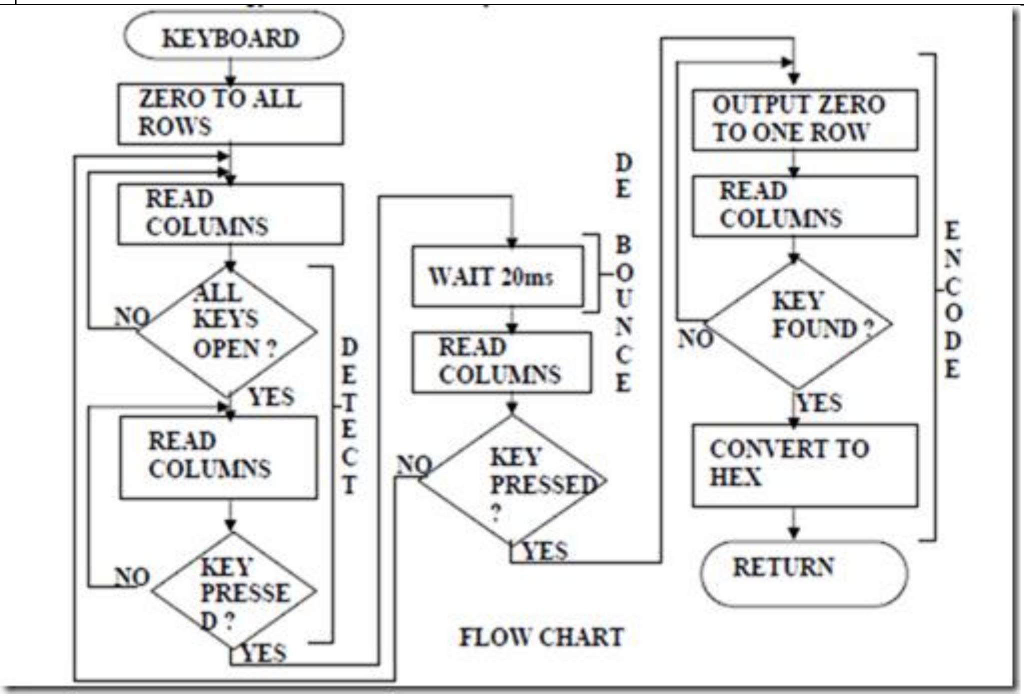


- **Mode 0 –**In this mode all the three ports (port A, B, C) can work as simple input function or simple output function. In this mode there is no interrupt handling capacity.
- **Mode 1 –** Handshake I/O mode or strobbed I/O mode. In this mode either port A or port B can work as simple input port or simple output port, and port C bits are used for handshake signals before actual data transmission. It has interrupt handling capacity and input and output are latched.
  Example: A CPU wants to transfer data to a printer. In this case since speed of processor is very fast as compared to relatively slow printer, so before actual data transfer it will send handshake signals to the printer for synchronization of the speed of the CPU and the peripherals.



- **Mode 2 –** Bi-directional data bus mode. In this mode only port A works, and port B can work either in mode 0 or mode 1. 6 bits port C are used as handshake signals. It also has interrupt handling capacity.

| | | | |
|---|---|---|---|
| 5. | a) | Explain how a matrix keyboard is interfaced with 8086 microprocessors with the help of a neat sketch. CO2 | 5M |



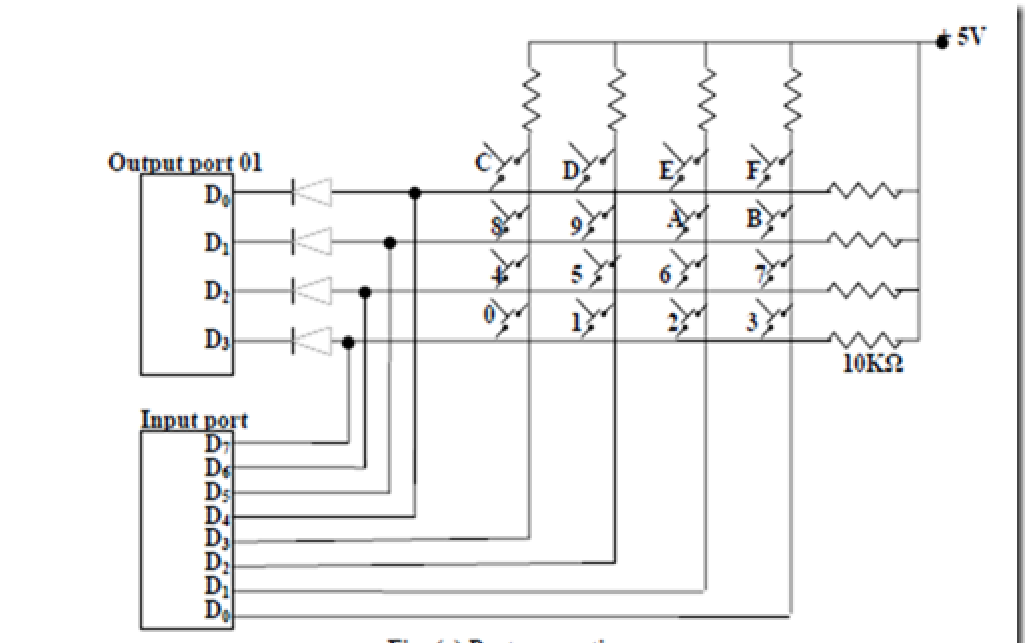


Fig: (a) Port connections

| | | | | |
|---|---|---|---|---|
| 5 | b) | Write in detail about modes of operation of 8254 | CO2 | 5M |

Operating modes

After power-up, the state of the 8254 is undefined. The Mode, count value, and output of all Counters are undefined. How each counter operates is determined when it is programmed. Each counter must be programmed before it can be used. Unused counters need not be programmed. Counters are programmed by writing a Control Word and then an initial count.

**Control Word of 8254 –**

The format of Control Word of 8254 is:

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| $SC_1$ | $SC_0$ | $RW_1$ | $RW_2$ | $M_2$ | $M_1$ | $M_0$ | BCD / Binary |

Here by using the value of SC1 and SC0 we select a specific counter:

| SC1 | SC0 | SELECTION |
|---|---|---|

| SC1 | SC0 | SELECTION |
|---|---|---|
| 0 | 0 | C0 |
| 0 | 1 | C1 |
| 1 | 0 | C2 |
| 1 | 1 | Read back status |

The values of RW1 and RW0 are used to decide the Read – Write operation:

| RW1 | RW0 | SELECTION |
|---|---|---|
| 0 | 0 | Counter Latch Command |
| 0 | 1 | Read/Write lower byte |
| 1 | 0 | Read/Write higher byte |
| 1 | 1 | Read/Write lower byte followed by higher byte |

The values of M2, M1, M0 are used to decide the operating modes of 8254:

| M2 | M1 | M0 | OPERATING MODE |
|---|---|---|---|
| 0 | 0 | 0 | MODE 0 |
| 0 | 0 | 1 | MODE 1 |
| X (0/1) | 1 | 0 | MODE 2 |
| X (0/1) | 1 | 1 | MODE 3 |
| 1 | 0 | 0 | MODE 4 |
| 1 | 0 | 1 | MODE 5 |

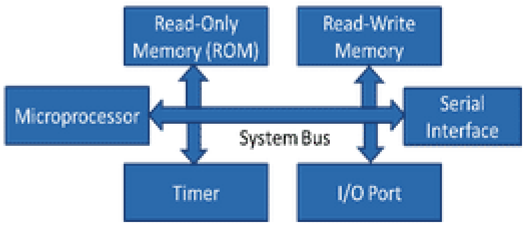
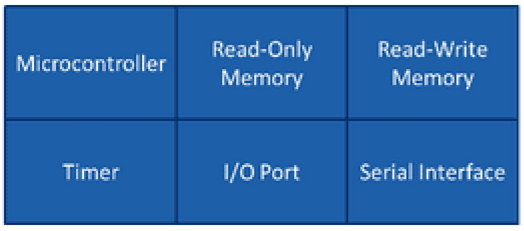
**Operating Modes of 8254:**

1. **Mode 0 (Interrupt on Terminal Count) –** Mode 0 is typically used for event counting. After the Control Word is written, OUT is initially low, and will remain low until the counter reaches zero it is decremented by 1 after every clock cycle. OUT then goes high and remains high until a new count or a new Mode 0 Control Word is written into the counter. GATE = 1 enables counting, GATE = 0 disables counting.
2. **Mode 1 (Hardware Retriggreable One Shot) –** OUT will be initially high. OUT will go low on the CLK pulse following a trigger to begin the one-shot pulse, and will remain low until the counter reaches zero.
3. **Mode 2 (Rate Generator) –** Initially value of OUT is low. When counting is enabled, it becomes high and this process repeats periodically. Value of count = Input Frequency / Output Frequency. This mode works as a frequency divider.
4. **Mode 3 (Square Wave Generator) –** Counting is enabled when GATE = 1 and disabled when GATE = 0. This mode is used to generate square waveform and time period (equal to count) is generated.

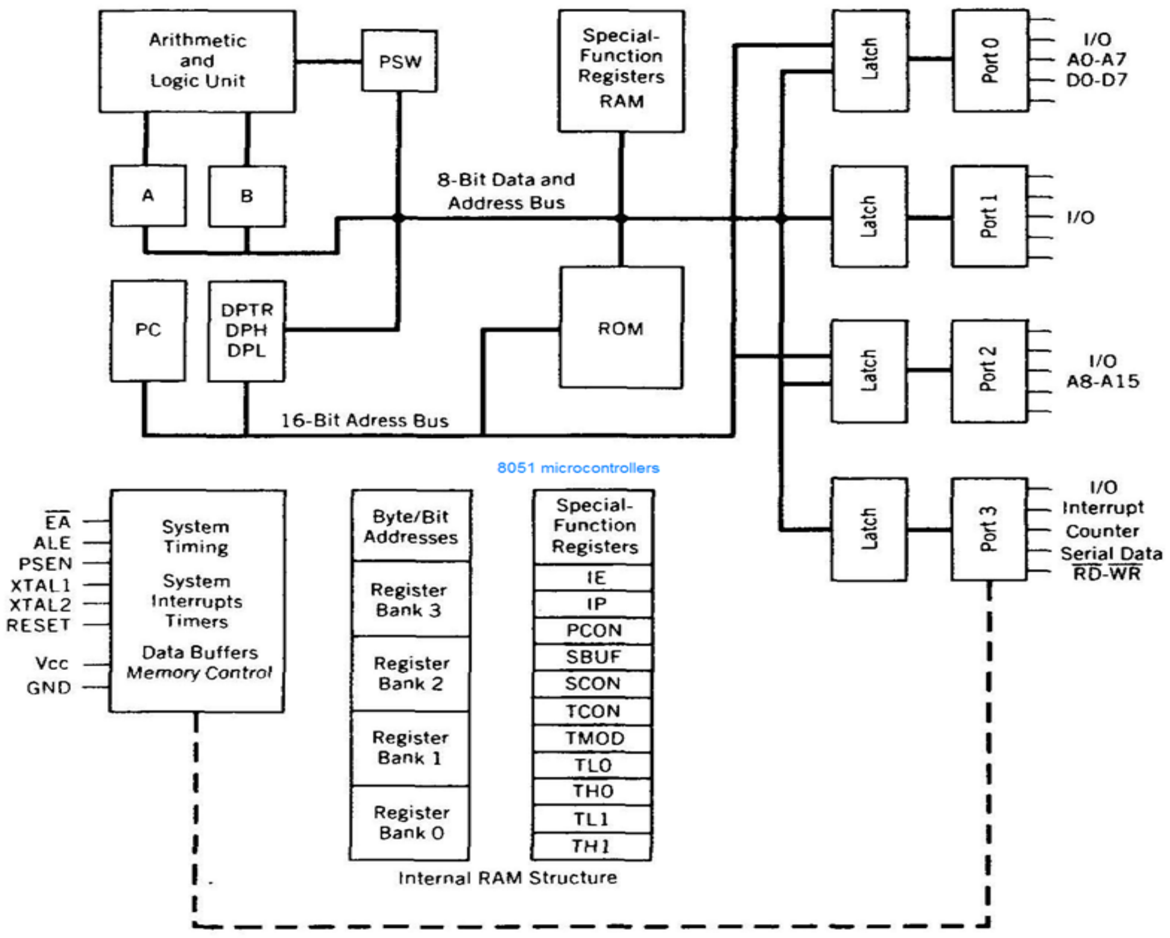
If N is count and is even then ontime of wave = N/2 and offtime = N/2
If N is odd the ontime = (N + 1) / 2 and offtime = (N – 1) / 2

5. **Mode 4 (Software Triggered Strobe) –** In this mode counting is enabled by using GATE = 1 and disabled by GATE = 0. Initially value of OUT is high and becomes low when value of count is at last stage. Count is reloaded again for subsequent clock pulse.
6. **Mode 5 (Hardware Triggered Strobe) –** OUT will initially be high. Counting is triggered by a rising edge of GATE. When the initial count has expired, OUT will go low for one clock pulse and then go high again. After writing the Control Word and initial count, the counter will not be loaded until the clock pulse after a trigger.

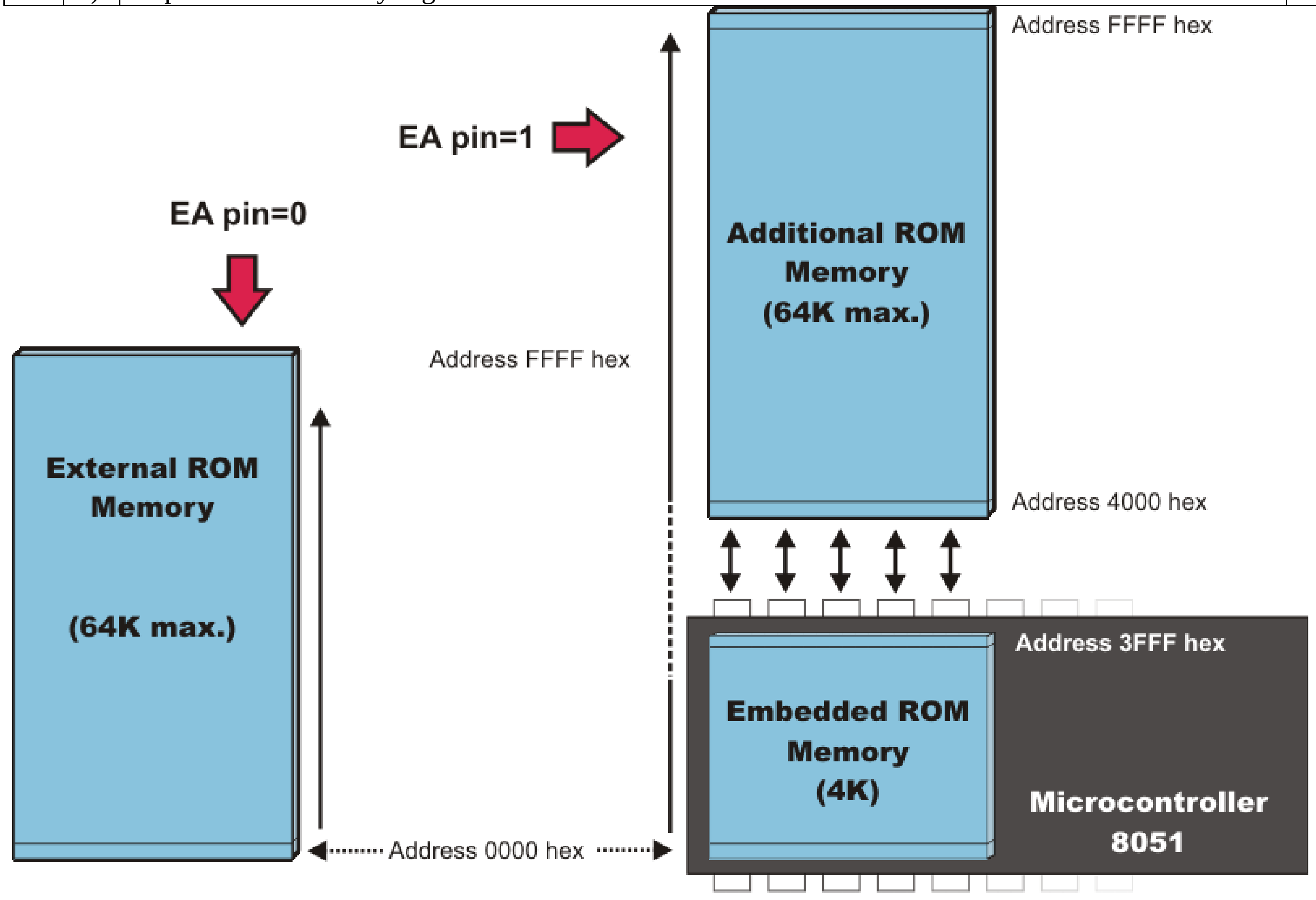
| | | | | |
|---|---|---|---|---|
| 6. | a) | Write about the comparison of microprocessors with microcontrollers | CO3 | 5M |

| Microprocessor | Micro Controller |
|---|---|
|   Read-Only Memory (ROM)<br>Read-Write Memory<br>Microprocessor<br>Serial Interface<br>System Bus<br>Timer<br>I/O Port |   Microcontroller<br>Read-Only Memory<br>Read-Write Memory<br>Timer<br>I/O Port<br>Serial Interface |
| Microprocessor is heart of Computer system. | Micro Controller is a heart of embedded system. |
| It is just a processor. Memory and I/O components have to be connected externally | Micro controller has external processor along with internal memory and i/O components |
| Since memory and I/O has to be connected externally, the circuit becomes large. | Since memory and I/O are present internally, the circuit is small. |
| Cannot be used in compact systems and hence inefficient | Can be used in compact systems and hence it is an efficient technique |
| Cost of the entire system increases | Cost of the entire system is low |
| Due to external components, the entire power consumption is high. Hence it is not suitable to used with devices running on stored power like batteries. | Since external components are low, total power consumption is less and can be used with devices running on stored power like batteries. |
| Most of the microprocessors do not have power saving features. | Most of the micro controllers have power saving modes like idle mode and power saving mode. This helps to reduce power consumption even further. |
| Since memory and I/O components are all external, each instruction will need external operation, hence it is relatively slower. | Since components are internal, most of the operations are internal instruction, hence speed is fast. |
| Microprocessor have less number of registers, hence more operations are memory based. | Micro controller have more number of registers, hence the programs are easier to write. |
| Microprocessors are based on von Neumann model/architecture where program and data are stored in same memory module | Micro controllers are based on Harvard architecture where program memory and Data memory are separate |
| Mainly used in personal computers | Used mainly in washing machine, MP3 players |

| | | | | |
|---|---|---|---|---|
| 6 | b) | Draw the internal architecture of 8051 and explain different blocks. | CO3 | 5M |



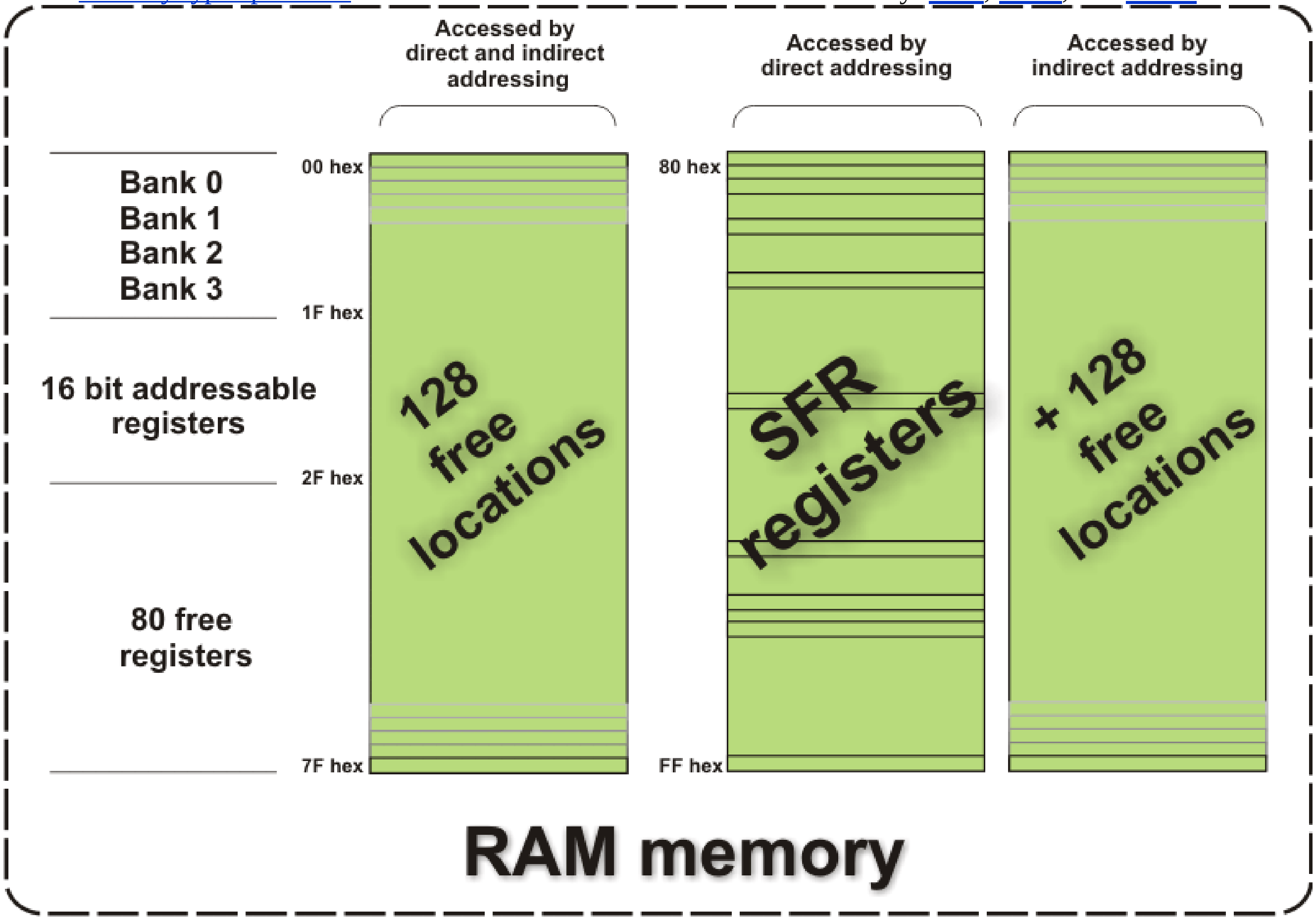
| | | | | |
|---|---|---|---|---|
| 7. | a) | Explain about memory organization of 8051 microcontrollers | CO3 | 5M |



## Internal Data Memory

Up to 256 bytes of internal data memory are available depending on the 8051 derivative. Locations available to the user occupy addressing space from 0 to 7Fh, i.e. first 128 registers and this part of RAM is divided in several blocks. The first 128 bytes of internal data memory are both directly and indirectly addressable. The upper 128 bytes of data memory (from 0x80 to 0xFF) can be addressed only indirectly.

Since internal data memory is used for CALL stack also and there is only 256 bytes splited over few different memory areas fine utilizing of this memory is crucial for fast and compact code. See types efficiency also.
Memory block in the range of 20h to 2Fh is bit-addressable, which means that each bit being there has its own address from 0 to 7Fh. Since there are 16 such registers, this block contains in total of 128 bits with separate addresses ( Bit 0 of byte 20h has the bit address 0, and bit 7 of byte 2Fh has the bit address 7Fh).
Three memory type specifiers can be used to refer to the internal data memory: **data**, **idata**, and **bdata**.



## External Data Memory

Access to external memory is slower than access to internal data memory. There may be up to 64K Bytes of external data memory. Several 8051 devices provide on-chip XRAM space that is accessed with the same instructions as the traditional external data space. This XRAM space is typically enabled via proper setting of SFR register and overlaps the external memory space. Setting of that register must be manualy done in code, before any access to external memory or XRAM space is made.
The mikroC PRO for 8051 has two memory type specifiers that refers to external memory space: **xdata** and **pdata**.

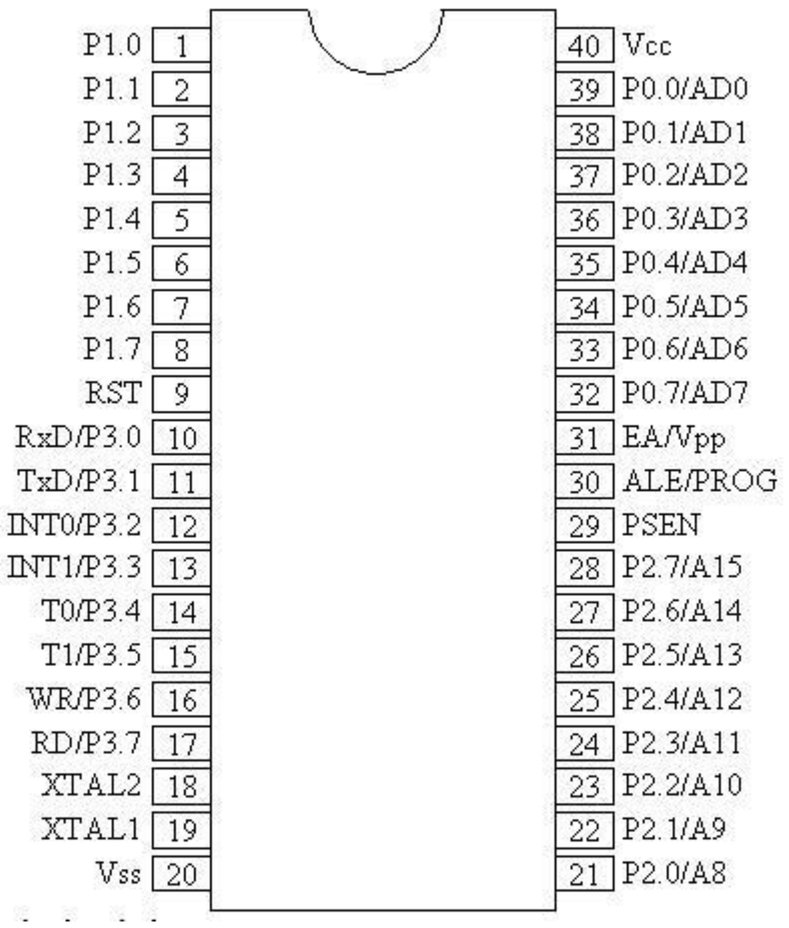
## SFR Memory

The 8051 provides 128 bytes of memory for Special Function Registers (SFRs). SFRs are bit, byte, or word-sized registers that are used to control timers, counters, serial I/O, port I/O, and peripherals.

| | | | | |
|---|---|---|---|---|
| 7 | b) | Explain about I/O port programming in 8051 microcontrollers | CO3 | 5M |

In 8051, I/O operations are done using four ports and 40 pins. The following pin diagram shows the details of the 40 pins. I/O operation port reserves 32 pins where each port has 8 pins. The other 8 pins are designated as $V_{cc}$, GND, XTAL1, XTAL2, RST, EA (bar), ALE/PROG (bar), and PSEN (bar).

It is a 40 Pin PDIP (Plastic Dual Inline Package)



**Note** − In a DIP package, you can recognize the first pin and the last pin by the cut at the middle of the IC. The first pin is on the left of this cut mark and the last pin (i.e. the 40th pin in this case) is to the right of the cut mark.

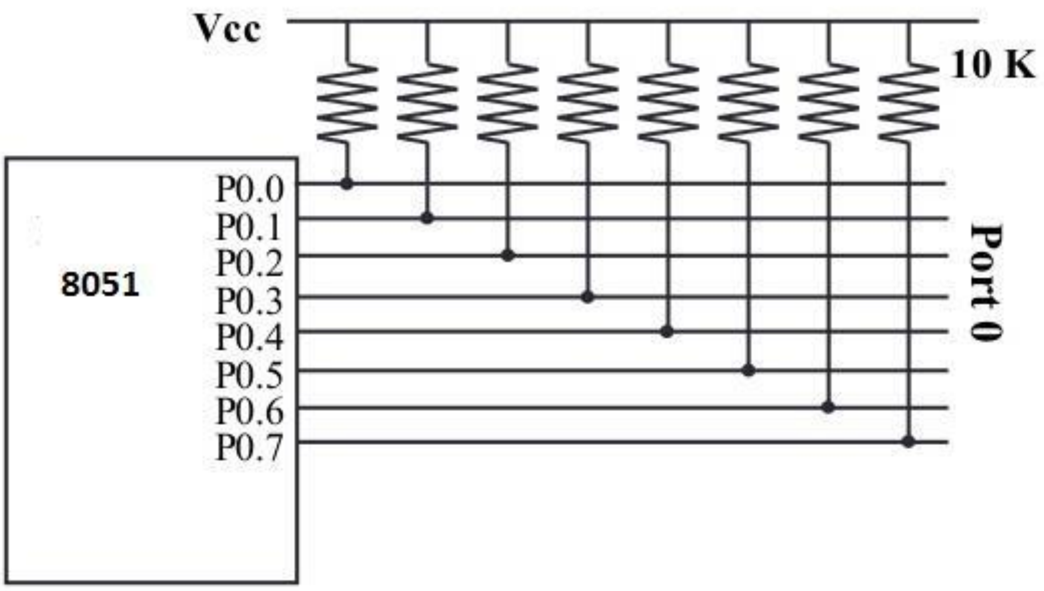
*I/O Ports and their Functions*

The four ports P0, P1, P2, and P3, each use 8 pins, making them 8-bit ports. Upon RESET, all the ports are configured as inputs, ready to be used as input ports. When the first 0 is written to a port, it becomes an output. To reconfigure it as an input, a 1 must be sent to a port.

Port 0 (Pin No 32 – Pin No 39)

It has 8 pins (32 to 39). It can be used for input or output. Unlike P1, P2, and P3 ports, we normally connect P0 to 10K-ohm pull-up resistors to use it as an input or output port being an open drain.

It is also designated as AD0-AD7, allowing it to be used as both address and data. In case of 8031 (i.e. ROMless Chip), when we need to access the external ROM, then P0 will be used for both Address and Data Bus. ALE (Pin no 31) indicates if P0 has address or data. When ALE = 0, it provides data D0-D7, but when ALE = 1, it has address A0-A7. In case no external memory connection is available, P0 must be connected externally to a 10K-ohm pull-up resistor.



```
MOV A,#0FFH  ;(comments: A=FFH(Hexadecimal  i.e. A=1111 1111)

MOV P0,A     ;(Port0 have 1's on every pin so that it works as Input)
```

Port 1 (Pin 1 through 8)

It is an 8-bit port (pin 1 through 8) and can be used either as input or output. It doesn't require pull-up resistors because they are already connected internally. Upon reset, Port 1 is configured as an input port. The following code can be used to send alternating values of 55H and AAH to Port 1.

```
;Toggle all bits of continuously
MOV     A,#55
BACK:

MOV     P2,A
ACALL   DELAY
CPL     A      ;complement(invert) reg. A
SJMP    BACK
```

If Port 1 is configured to be used as an output port, then to use it as an input port again, program it by writing 1 to all of its bits as in the following code.

```
;Toggle all bits of continuously

MOV     A ,#0FFH    ;A = FF hex
MOV     P1,A        ;Make P1 an input port
MOV     A,P1        ;get data from P1
MOV     R7,A        ;save it in Reg R7
ACALL   DELAY       ;wait

MOV     A,P1        ;get another data from P1
MOV     R6,A        ;save it in R6
ACALL   DELAY       ;wait

MOV     A,P1        ;get another data from P1
MOV     R5,A        ;save it in R5
```

Port 2 (Pins 21 through 28)

Port 2 occupies a total of 8 pins (pins 21 through 28) and can be used for both input and output operations. Just as P1 (Port 1), P2 also doesn't require external Pull-up resistors because they are already connected internally. It must be used along with P0 to provide the 16-bit address for the external memory. So it is also designated as (A0–A7), as shown in the pin diagram. When the 8051 is connected to an external memory, it provides path for upper 8-bits of 16-

bits address, and it cannot be used as I/O. Upon reset, Port 2 is configured as an input port. The following code can be used to send alternating values of 55H and AAH to port 2.

```
;Toggle all bits of continuously
MOV     A,#55
BACK:
MOV     P2,A
ACALL   DELAY
CPL     A        ; complement(invert) reg. A
SJMP    BACK
```

If Port 2 is configured to be used as an output port, then to use it as an input port again, program it by writing 1 to all of its bits as in the following code.

```
;Get a byte from P2 and send it to P1
MOV    A,#0FFH    ;A = FF hex
MOV    P2,A       ;make P2 an input port
BACK:
MOV    A,P2       ;get data from P2
MOV    P1,A       ;send it to Port 1
SJMP   BACK        ;keep doing that
```

Port 3 (Pins 10 through 17)

It is also of 8 bits and can be used as Input/Output. This port provides some extremely important signals. P3.0 and P3.1 are RxD (Receiver) and TxD (Transmitter) respectively and are collectively used for Serial Communication. P3.2 and P3.3 pins are used for external interrupts. P3.4 and P3.5 are used for timers T0 and T1 respectively. P3.6 and P3.7 are Write (WR) and Read (RD) pins. These are active low pins, means they will be active when 0 is given to them and these are used to provide Read and Write operations to External ROM in 8031 based systems.

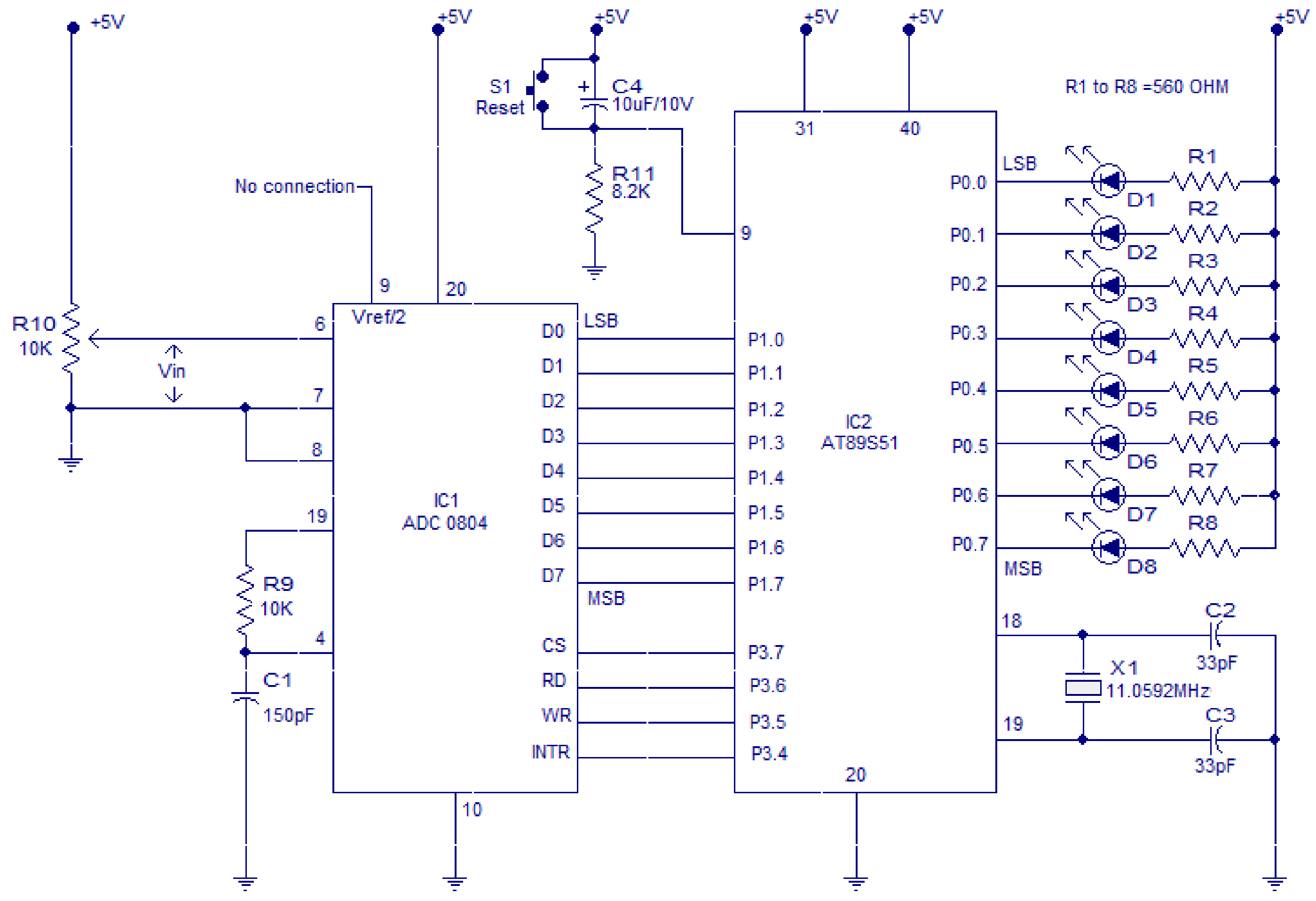
| **P3 Bit** | **Function** | **Pin** |
|---|---|---|
| P3.0 | RxD | 10 |
| P3.1 < | TxD | 11 |
| P3.2 < | Complement of INT0 | 12 |
| P3.3 < | INT1 | 13 |
| P3.4 < | T0 | 14 |
| P3.5 < | T1 | 15 |
| P3.6 < | WR | 16 |
| P3.7 < | Complement of RD | 17 |

*Dual Role of Port 0 and Port 2*

- **Dual role of Port 0** − Port 0 is also designated as AD0–AD7, as it can be used for both data and address handling. While connecting an 8051 to external memory, Port 0 can provide both address and data. The 8051 microcontroller then multiplexes the input as address or data in order to save pins.

- **Dual role of Port 2** − Besides working as I/O, Port P2 is also used to provide 16-bit address bus for external memory along with Port 0. Port P2 is also designated as (A8– A15), while Port 0 provides the lower 8-bits via A0–A7. In other words, we can say that when an 8051 is connected to an external memory (ROM) which can be maximum up to 64KB and this is possible by 16 bit address bus because we know 216 = 64KB. Port2 is used for the upper 8-bit of the 16 bits address, and it cannot be used for I/O and this is the way any Program code of external ROM is addressed.

| | | | | |
|---|---|---|---|---|
| 8. | a) | Explain about interfacing of ADC with 8051 microcontrollers using a diagram | CO4 | 5M |



ADC (Analog to digital converter) forms a very essential part in many embedded projects and this article is about interfacing an ADC to 8051 embedded controller. ADC 0804 is the ADC used here and before going through the interfacing procedure, we must neatly understand how the ADC 0804 works.

**Steps for converting the analogue input and reading the output from ADC0804.**

- Make CS=0 and send a low to high pulse to WR pin to start the conversion.
- Now keep checking the INTR pin. INTR will be 1 if conversion is not finished and INTR will be 0 if conversion is finished.
- If conversion is not finished (INTR=1) , poll until it is finished.
- If conversion is finished (INTR=0), go to the next step.
- Make CS=0 and send a high to low pulse to RD pin to read the data from the ADC

ADC0804 is an 8 bit successive approximation analogue to digital converter from National semiconductors. The features of ADC0804 are differential analogue voltage inputs, 0-5V input voltage range, no zero adjustment, built in clock generator, reference voltage can be externally adjusted to convert smaller analogue voltage span to 8 bit resolution etc. The pin out diagram of ADC0804 is shown in the figure below.
The figure above shows the schematic for interfacing ADC0804 to 8051. The circuit initiates the ADC to convert a given analogue input , then accepts the corresponding digital data and displays it on the LED array connected at P0. For example, if the analogue input voltage Vin is 5V then all LEDs will glow indicating 11111111 in binary which is the equivalent of 255 in decimal. AT89s51 is the microcontroller used here. Data out pins (D0 to D7) of the ADC0804

are connected to the port pins P1.0 to P1.7 respectively. LEDs D1 to D8 are connected to the port pins P0.0 to P0.7 respectively. Resistors R1 to R8 are current limiting resistors. In simple words P1 of the microcontroller is the input port and P0 is the output port. Control signals for the ADC (INTR, WR, RD and CS) are available at port pins P3.4 to P3.7 respectively. Resistor R9 and capacitor C1 are associated with the internal clock circuitry of the ADC. Preset resistor R10 forms a voltage divider which can be used to apply a particular input analogue voltage to the ADC. Push button S1, resistor R11 and capacitor C4 forms a debouncing reset mechanism. Crystal X1 and capacitors C2,C3 are associated with the clock circuitry of the microcontroller.

| | | | | |
|---|---|---|---|---|
| 8 | b) | Explain about timer port programming in 8051 microcontrollers | CO4 | 5M |

The programming of 8051 Timers can be done by using either polling method or by using interrupt. In polling, the microcontroller keeps monitoring the status of Timer flag. While doing so, it does no other operation and consumes all its processing time in checking the Timer flag until it is **raised on a rollover**. In interrupt method controller responds to only when the Timer flag is raised. The interrupt method prevents the wastage of controller's processing time unlike polling method.
Polling is mostly used for time delay generation and interrupt method is more useful when waveforms are to be generated or some action has to be repeated in fixed delays.

**(i) Polling Method**

The polling method involves the following algorithm:

1. Configure the Timer mode by passing a hex value into the TMOD register. This will tell the controller about which Timer is be used; the mode of Timer; operation (to be used as timer or counter); and whether external interrupt is required to start Timer.

2. Load the initial values in the Timer low TLx and high THx byte. (x = 0/1)

3. Start the Timer by setting TRx bit.

4. Wait while the Timer flag TFx is raised.

5. Clear the Timer flag. The Timer flag is raised when Timer **rolls over** from FFFFH to 0000H. If the Timer is not stopped, it will start updating from 0000H in case of modes 0 & 1 while with initial value in case of mode 2. If TFx is not cleared, controller will not be able to detect next rollover.

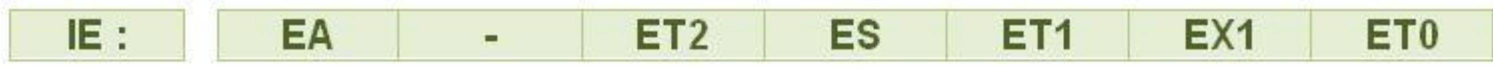
6. Stop the Timer by clearing TRx bit. If TRx bit is not cleared the Timer will restart updating from 0000H after the rollover in case of modes 0 and 1 while with initial value in case of mode 2.

**(ii) Interrupt Method**

The interrupt method makes use of a register called **Interrupt Enable** (IE) register. An 8051 microcontroller has 6 hardware interrupts. The interrupts refer to a notification, communicated to the controller, by a hardware device or software, on receipt of which controller skips temporarily whatsoever it was doing and responds to the interrupt.

The controller starts the execution of an **Interrupt Service Routine** (ISR) or Interrupt Handler which is a piece of code that tells the processor or controller **what to do on receipt of an interrupt**. After the execution of ISR, controller returns to whatever it was doing earlier (before the interrupt was received).

The Interrupt Enable register has the following bits which enable or disable the hardware interrupts of 8051 microcontroller.



| IE : | EA | - | ET2 | ES | ET1 | EX1 | ET0 |
|---|---|---|---|---|---|---|---|

*Fig. 4: Bit Values of IE Register of 8051 Microcontroller*

When EA (IE.7) is set (=1), interrupts are enabled. When clear (EA=0), interrupts are disabled and controller does not respond to any interrupts.

ET0, ET1 & ET2 (IE.3, IE.4 & IE.5) are Timer interrupt bits. When set (high) the timer are enabled and when cleared (low) they are disabled. (8052 controllers have three Timers, so ET2 is its Timer 2 interrupt bit.) The ISR corresponding to these interrupts are executed when the TFx flags of respective Timers are raised. For more details on other IE register bits, refer Interrupts with 8051.

Note that IE register is bit addressable.

Timer programming using Timer interrupts involves following algorithm.

1. Configure the Timer mode by passing a hex value to TMOD register.

2. Load the initial values in the Timer low TLx and high THx byte.

3. Enable the Timer interrupt by passing hex value to IE register or setting required bits of IE register. For example,

   *IE = 0x82;* enables Timer 0 interrupt

   *IE = 0x88;* enables Timer 1 interrupt

   or

   *EA = 1;*

   *ET0 = 1;* enables Timer 0 interrupt

   *IE^7 = 1;*

   *IE^3 = 1;* enables Timer 1 interrupt

4. Start the Timer by setting TRx bit.

5. Write Interrupt Service Routine (ISR) for the Timer interrupt. For example,

   ISR definition for Timer 0 :

```
void ISR_Timer0(void) interrupt 1
{
<Body of ISR>
}
```

   ISR definition for Timer 1 :

```
void ISR_Timer1(void) interrupt 3
{
<Body of ISR>
}
```

6. If the Timer has to be stopped after once the interrupt has occurred, the ISR must contain the statement to stop the Timer.

   For example,

```
void ISR_Timer1(void) interrupt 3
{
<Body of ISR>
TR1 =0;
}
```

7. If a routine written for Timer interrupt has to be repeated again and again, the Timer run bit need not be cleared. But it should be kept in mind that **Timer will start updating from 0000H** and **not the initial values in case of mode 0 and 1**. So the initial values must be reloaded in the interrupt service routine.

**Different modes of a Timer**

There are four Timer modes designated as Modes 0, 1, 2 and 3. A particular mode is selected by configuring the M1 & M0 bits of TMOD register.

| Mode | M1 | M0 | Operation |
|---|---|---|---|
| **Mode 0** | 0 | 0 | 13-bit Timer |
| **Mode 1** | 0 | 1 | 16-bit Timer |
| **Mode 2** | 1 | 0 | 8-bit Auto Reload |
| **Mode 3** | 1 | 1 | Split Timer Mode |

**(i) Mode 0 : 13-bit Timer**

Mode 0 is a 13 bit Timer mode and uses 8 bits of high byte and 5 bit prescaler of low byte. The value that the Timer can update in mode0 is from 0000H to 1FFFH. The 5 bits of lower byte append with the bits of higher byte. The Timer rolls over from 1FFFH to 0000H to raise the Timer flag.

**(ii) Mode 1 : 16-bit Timer**

Mode1 is one of the most commonly used Timer modes. It allows all 16 bits to be used for the Timer and so it allows values to vary from 0000H to FFFFH.

If a value, say YYXXH, is loaded into the Timer bytes, then the delay produced by the Timer will be equal to the product :

[ ( **FFFFH – YYXXH +1** ) x ( **period of one timer clock** ) ].

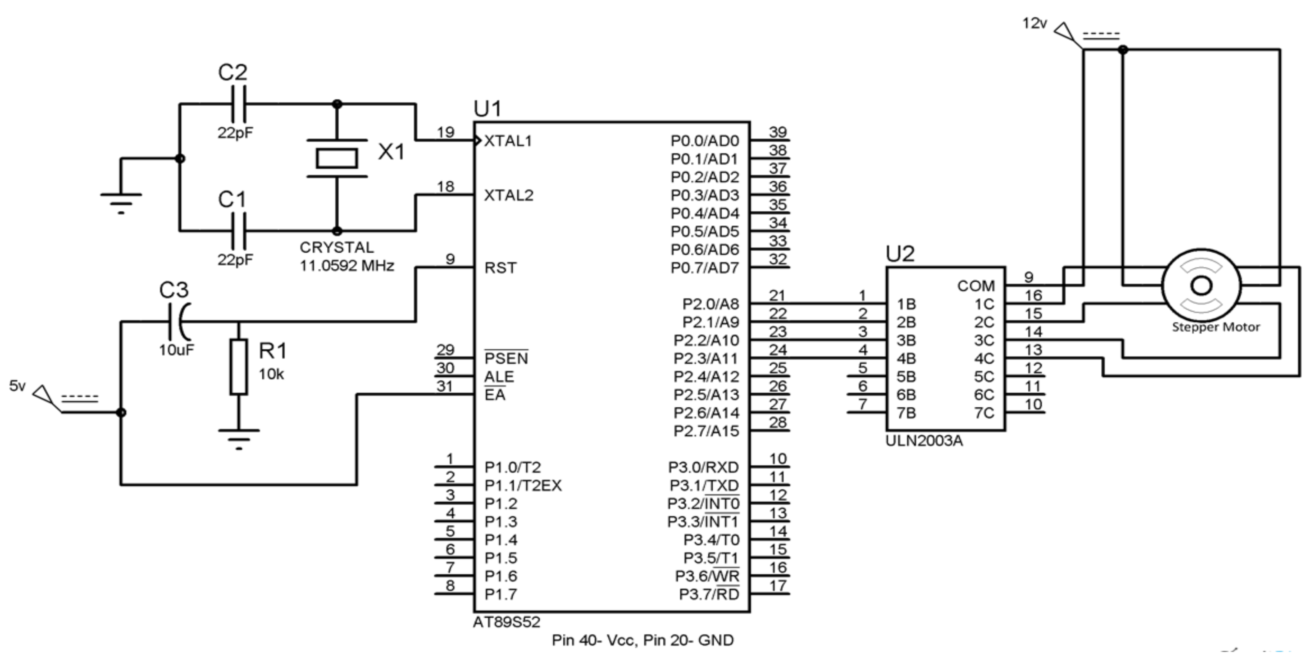
It can also be considered as follows: convert YYXXH into decimal, say NNNNN, then delay will be equal to the product :

[ ( **65536-NNNNN** ) x ( **period of one timer clock** ) ].

The period of one timer clock is 1.085 µs for a crystal of 11.0592 MHz frequency as discussed above.

Now **to produce a desired delay**, divide the required delay by the Timer clock period. Assume that the division yields a number NNNNN. This is the number of times Timer must be updated before it stops. Subtract this number from 65536 (binary equivalent of FFFFH) and convert the difference into hex. This will be the initial value to be loaded into the Timer to get the desired delay.

| | | | | |
|---|---|---|---|---|
| 9. | a) | Explain how a stepper motor is interfaced to an 8051 µc with a neat diagram. | CO4 | 5M |



Interfacing with 8051 is very easy we just need to give the 0 and 1 to the four wires of stepper motor according to the above tables depending on which mode we want to run the stepper motor. And rest two wires should be connected to a proper 12v supply (depending on the stepper motor). Here we have used the unipolar stepper motor. We have connected four ends of the coils to the first four pins of port 2 of 8051 through the ULN2003A.

nterfacing with 8051 is very easy we just need to give the 0 and 1 to the four wires of stepper motor according to the above tables depending on which mode we want to run the stepper motor. And rest two wires should be connected to a proper 12v supply (depending on the stepper motor). Here we have used the unipolar stepper motor. We have connected four ends of the coils to the first four pins of port 2 of 8051 through the ULN2003A.

**Stepper motors** are basically two types: Unipolar and Bipolar. **Unipolar stepper** motor generally has five or six wire, in which four wires are one end of four stator coils, and other end of the all four coils is tied together which represents fifth wire, this is called common wire (common point). Generally there are two common wire, formed by connecting one end of the two-two coils as shown in below figure. Unipolar stepper motor is very common and popular because of its ease of use.

In **Bipolar stepper** motor there is just four wires coming out from two sets of coils, means there are no common wire.

Stepper motor is made up of a stator and a rotator. Stator represents the four electromagnet coils which remain stationary around the rotator, and rotator represents permanent magnet which rotates. Whenever the coils energised by applying the current, the electromagnetic field is created, resulting the rotation of rotator (permanent magnet). Coils should be energised in a particular sequence to make the rotator rotate. On the basis of this “sequence” we can divide the working method of **Unipolar stepper motor** in three modes: Wave drive mode, full step drive mode and half step drive mode.

**Wave drive mode**: In this mode one coil is energised at a time, all four coil are energised one after another. It produces less torque in compare with Full step drive mode but power consumption is less. Following is the table for producing this mode using microcontroller, means we need to give Logic 1 to the coils in the sequential manner.

| **Steps** | **A** | **B** | **C** | **D** |
|---|---|---|---|---|
| **1** | 1 | 0 | 0 | 0 |
| **2** | 0 | 1 | 0 | 0 |
| **3** | 0 | 0 | 1 | 0 |
| **4** | 0 | 0 | 0 | 1 |

**Full Drive mode:** In this, two coil are energised at the same time producing high torque. Power consumption is higher. We need to give Logic 1 to two coils at the same time, then to the next two coils and so on.

| **Steps** | **A** | **B** | **C** | **D** |
|---|---|---|---|---|
| **1** | 1 | 1 | 0 | 0 |
| **2** | 0 | 1 | 1 | 0 |
| **3** | 0 | 0 | 1 | 1 |
| **4** | 1 | 0 | 0 | 1 |

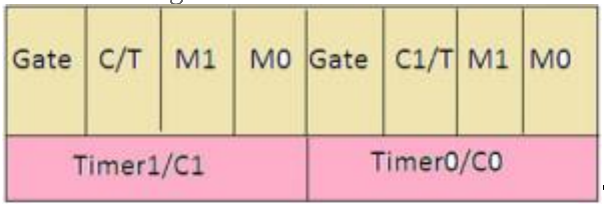
**Half Drive mode:** In this mode one and two coils are energised alternatively, means firstly one coil is energised then two coils are energised then again one coil is energised then again two, and so on. This is combination of full and wave drive mode, and used to increase the angular rotation of the motor.

| **Steps** | **A** | **B** | **C** | **D** |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| **1** | 1 | 0 | 0 | 0 |
| **2** | 1 | 1 | 0 | 0 |
| **3** | 0 | 1 | 0 | 0 |
| **4** | 0 | 1 | 1 | 0 |
| **5** | 0 | 0 | 1 | 0 |
| **6** | 0 | 0 | 1 | 1 |
| **7** | 0 | 0 | 0 | 1 |
| **8** | 1 | 0 | 0 | 1 |

| 9 | b) | Draw the format of (i) TMOD (ii) SCON special function registers in 8051 microcontrollers. | CO4 | 5M |
|---|---|---|---|---|

**(i) Timer Mode Control (TMOD):** TMOD is an 8-bit register used for selecting timer or counter and mode of timers. Lower 4-bits are used for control operation of timer 0 or counter0, and remaining 4-bits are used for control operation of timer1 or counter1.This register is present in SFR register, the address for SFR register is 89th.



| Gate | C/T | M1 | M0 | Gate | C1/T | M1 | M0 |
|---|---|---|---|---|---|---|---|
| Timer1/C1 | | | | Timer0/C0 | | | |

Timer Mode Control (TMOD)

**Gate:** If the gate bit is set to '0', then we can start and stop the "software" timer in the same way. If the gate is set to '1', then we can perform hardware timer.

**C/T:** If the C/T bit is '1', then it is acting as a counter mode, and similarly when set C+ =/T bit is '0'; it is acting as a timer mode.

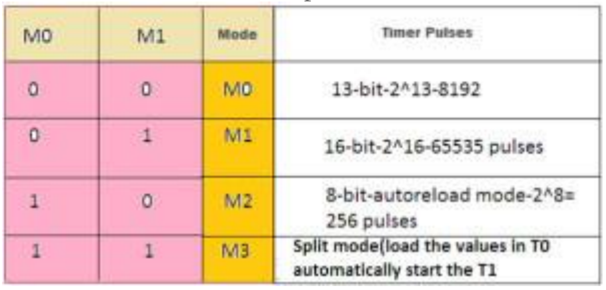
**Mode select bits:** The M1 and M0 are mode select bits, which are used to select the timer operations. There are four modes to operate the timers.

**Mode 0:** This is a 13-bit mode that means the timer operation completes with "8192" pulses.

**Mode 1:** This is a16-bit mode, which means the timer operation completes with maximum clock pulses that "65535".

**Mode 2:** This mode is an 8-bit auto reload mode, which means the timer operation completes with only "256" clock pulses.

**Mode 3:** This mode is a split-timer mode, which means the loading values in T0 and automatically starts the T1.

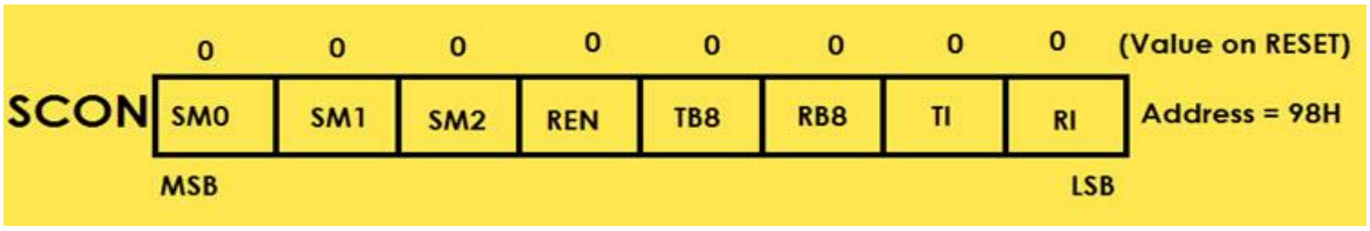


| M0 | M1 | Mode | Timer Pulses |
|---|---|---|---|
| 0 | 0 | M0 | 13-bit-2^13-8192 |
| 0 | 1 | M1 | 16-bit-2^16-65535 pulses |
| 1 | 0 | M2 | 8-bit-autoreload mode-2^8= 256 pulses |
| 1 | 1 | M3 | Split mode(load the values in T0 automatically start the T1 |

**(ii)SCON (Serial Control)**

The Serial Control or SCON SFR is used to control the 8051 Microcontroller's Serial Port. It is located as an address of 98H. Using SCON, you can control the Operation Modes of the Serial Port, Baud Rate of the Serial Port and Send or Receive Data using Serial Port.

SCON Register also consists of bits that are automatically SET when a byte of data is transmitted or received.



SCON
0 0 0 0 0 0 0 0 (Value on RESET)
SM0 SM1 SM2 REN TB8 RB8 TI RI
Address = 98H
MSB
LSB

**Serial Port Mode Control Bits**

| SM0 | SM1 | Mode | Description | Baud Rate |
|---|---|---|---|---|
| 0 | 0 | 0 | 8-Bit Synchronous Shift Register Mode | Fixed Baud Rate ( Frequency of oscillator / 12) |
| 0 | 1 | 1 | 8-bit Standard UART mode | Variable Baud Rate (Can be set by Timer 1) |
| 1 | 0 | 2 | 9-bit Multiprocessor Comm. mode | Fixed Baud Rate ( Frequency of oscillator / 32 or Frequency of oscillator / 64 |
| 1 | 1 | 3 | 9-bit Multiprocessor Comm. mode | Variable Baud Rate (Can be set by Timer 1) |

Dr HOD
[illegible]
4/3/2021

Prepared by
[illegible]
[illegible]
B. Praveen kumar.